KNN Find k examples $\left\{\mathbf{x}^{(i)}, t^{(i)}\right\}$ closest to the test instance $\mathbf{x}$ and then output majority $\arg\max_{t^z} \sum_{r=1}^{k} \delta(t^{(z)}, t^{(r)})$. Define $\delta(a,b)=1$ if $a=b$, 0 otw. **Choice of k:** Rule is $k<\sqrt{n}$, small k may overfit, while large may under-fit. **Curse of Dim:** In high dimensions, "most" points are approximately the same distance. **Computation Cost:** 0 (minimal) at training/ no learning involved. Query time find N distances in D dimension $\mathcal{O}(ND)$ and $\mathcal{O}(N\log N)$ sorting time.

Entropy $H(X) = -\mathbb{E}_{X\sim p}\left[\log_2 p(X)\right] = -\sum_{x\in X} p(x)\log_2 p(x)$ **Multi-class:** $H(X,Y) = -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(x,y)$ **Properties:** H is non-negative, $H(Y|X)\le H(Y)$, $X\perp Y \implies H(Y|X)=H(Y)$, $H(Y|Y)=0$, and $H(X,Y)=H(X|Y)+H(Y)=H(Y|X)+H(X)$

Expected Conditional Entropy $H(Y|X) = \mathbb{E}_{X\sim p(x)}[H(Y|X)] = \sum_{x\in X} p(x) H(Y|X=x) = -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(y|x) = -\mathbb{E}_{(X,Y)\sim p(x,y)}\left[\log_2 p(Y|X)\right]$ **Information Gain** $IG(Y|X) = H(Y)-H(Y|X)$

Bias Variance Decomposition Using the square error loss $L(y,t)=\frac{1}{2}(y-t)^2$, **Bias ($\uparrow\implies$ under-fitting):** How close is our classifier to true target. **Variance ($\uparrow\implies$ overfitting):** How widely dispersed are out predictions as we generate new datasets

$$\begin{aligned}
\mathbb{E}_{\mathbf{x},\mathcal{D}}\left[\left(h_{\mathcal{D}}(\mathbf{x})-t\right)^2\right] &= \mathbb{E}_{\mathbf{x},\mathcal{D}}\left[\left(h_{\mathcal{D}}(\mathbf{x})-\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]+\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]-t\right)^2\right]\\
&= \mathbb{E}_{\mathbf{x},\mathcal{D}}\left[\left(h_{\mathcal{D}}(\mathbf{x})-\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]\right)^2+\left(\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]-t\right)^2+2\left(h_{\mathcal{D}}(\mathbf{x})-\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]\right)\left(\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]-t\right)\right]\\
&= \underbrace{\mathbb{E}_{\mathbf{x},\mathcal{D}}\left[\left(h_{\mathcal{D}}(\mathbf{x})-\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]\right)^2\right]}_{\text{variance}}+\underbrace{\mathbb{E}_{\mathbf{x}}\left[\left(\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]-t\right)^2\right]}_{\text{bias}}
\end{aligned}$$

Bagging with Generating Distribution Suppose we could sample m independent training sets $\{\mathcal{D}_i\}_{i=1}^m$ from $p_{dataset}$. Learn $h_i := h_{\mathcal{D}_i}$ and out final predictor is $h = 1/m\sum_{i=1}^m h_i$. **Bias Unchanged:** $\mathbb{E}_{\mathcal{D}_1,\ldots,\mathcal{D}_m \overset{iid}{\sim} p_{\text{dataset}}}\left[h(\mathbf{x})\right] = \frac{1}{m}\sum_{i=1}^m \mathbb{E}_{\mathcal{D}_i\sim p_{\text{dataset}}}\left[h_i(\mathbf{x})\right] = \mathbb{E}_{\mathcal{D}\sim p_{\text{dataset}}}\left[h_{\mathcal{D}}(\mathbf{x})\right]$ **Variance Reduced:** $\text{Var}_{\mathcal{D}_1,\ldots,\mathcal{D}_m}[h(\mathbf{x})] = \frac{1}{m^2}\sum_{i=1}^m \text{Var}\left[h_i(\mathbf{x})\right] = \frac{1}{m}\text{Var}\left[h_{\mathcal{D}}(\mathbf{x})\right]$

Bootstrap Aggregation Take a single dataset $\mathcal{D}$ with n sample and generate m new datasets, each by sampling n training examples from $\mathcal{D}$, with replacement. We then the average the predictions. We have the reduction in variance to be $\text{Var}\left(\frac{1}{m}\sum_{i=1}^m h_i(\mathbf{x})\right) = \frac{1}{m}(1-\rho)\sigma^2+\rho\sigma^2$

Random Forest Upon bootstrap aggregation, for each bag we choose a random set of features to make the trees grow on (decorrelates predictions, lower ρ).

Bayes Optimality $\mathbb{E}_{\mathbf{x},\mathcal{D},t|\mathbf{x}}\left[\left(h_{\mathcal{D}}(\mathbf{x})-t\right)^2\right] = \underbrace{\mathbb{E}_{\mathbf{x}}\left[\left(\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]-y_*(\mathbf{x})\right)^2\right]}_{\text{bias}}+\underbrace{\mathbb{E}_{\mathbf{x},\mathcal{D}}\left[\left(h_{\mathcal{D}}(\mathbf{x})-\mathbb{E}_{\mathcal{D}}\left[h_{\mathcal{D}}(\mathbf{x})\right]\right)^2\right]}_{\text{variance}}+\underbrace{\mathbb{E}_{\mathbf{x}}[\text{Var}[t|\mathbf{x}]]}_{\text{Bayes}}$

Feature Mapping Some time we want fit a polynomial curve, we can do this using a feature map $y=\mathbf{w}^\top\boldsymbol{\psi}(x)$ where $\boldsymbol{\psi}(x) = \left[1,x,x^2,\ldots\right]^\top$. In general the feature map could be anything.

Ridge Regression $\mathbf{w}_\lambda^{Ridge} = \underset{\mathbf{w}}{\text{argmin}}\mathcal{J}_{\text{reg}}(\mathbf{w}) = \underset{\mathbf{w}}{\text{argmin}}\frac{1}{2}\|\mathbf{X}\mathbf{w}-\mathbf{t}\|_2^2+\frac{\lambda}{2}\|\mathbf{w}\|_2^2 = \left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{t}$ When $\lambda=0$ this is just OLS.

Gradient Descent Consider the some cost function $\mathcal{J}$ and we want to optimize it.

- **GD:** $\mathbf{w}\leftarrow\mathbf{w}-\alpha\frac{\partial\mathcal{J}}{\partial\mathbf{w}}$; **GD w/ Reg** $\mathbf{w}\leftarrow\mathbf{w}-\alpha\left(\frac{\partial\mathcal{J}}{\partial\mathbf{w}}+\lambda\frac{\partial\mathcal{R}}{\partial\mathbf{w}}\right) = (1-\alpha\lambda)\mathbf{w}-\alpha\frac{\partial\mathcal{J}}{\partial\mathbf{w}}$
- **mSGD:** Choose mini batch $\mathcal{M}\subset\{1,\ldots,N\}$ and update $\mathbf{w}\leftarrow\mathbf{w}-\frac{\alpha}{|\mathcal{M}|}\sum_{i=1}^{|\mathcal{M}|}\frac{\partial\mathcal{L}^{(i)}}{\partial\mathbf{w}}$ **Reasonable size** would be $|\mathcal{M}|\approx 100$
- **SGD:** Choose i at uniform; $\mathbf{w}\leftarrow\mathbf{w}-\alpha\frac{\partial\mathcal{L}^{(i)}}{\partial\mathbf{w}}$; **Pro//Cons:** Progress w/o seeing all data//High Variance & Not efficiently vectorized

Cross Entropy Loss $\mathcal{L}_{CE} = -t\log y-(1-t)\log(1-y)$ **Logistic CE** $\mathcal{L}_{LCE}(z,t) = \mathcal{L}_{CE}(\sigma(z),t) = t\log(1+e^{-z})+(1-t)\log(1+e^{z})$

Multi-class Classification

- **Softmax Function** Natural generalization of logistic func: $y_k = \text{softmax}\left(z_1,\ldots,z_K\right)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$; iuputs z_k are called logits.
- **CE Loss, Vectorized** $\mathcal{L}_{\text{CE}}(\mathbf{y},\mathbf{t}) = -\sum_{k=1}^K t_k\log y_k = -\mathbf{t}^\top(\log\mathbf{y})$ where the log is applied elementwise.
- **Softmax Regression** $\mathbf{z}=\mathbf{W}\mathbf{x}+\mathbf{b}$, $\mathbf{y}=\text{softmax}(\mathbf{z})$, and $\mathcal{L}_{CE} = -\mathbf{t}^\top(\log\mathbf{y})$; GD Updates is $\mathbf{w}_k\leftarrow\mathbf{w}_k-\alpha\frac{1}{N}\sum_{i=1}^N\left(y_k^{(i)}-t_k^{(i)}\right)\mathbf{x}^{(i)}$ where $\mathbf{w}_k$ means the k-th row of $\mathbf{W}$

Activation Functions **Identity** $y=z$ **ReLU** $y=\max(0,z)$ **Soft ReLU** $y=\log(1+e^z)$ **Thresholding** $y=1$ if $z>0$ else 0. **Logistic** $y=\frac{1}{1+e^{-z}}$ **tanh** $y=\frac{e^z-e^{-z}}{e^z+e^{-z}}$

Multilayer Perceptron

- **Modularity of Layers** $\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x}) = \phi\left(\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right)$, $\mathbf{h}^{(2)} = f^{(2)}\left(\mathbf{h}^{(1)}\right) = \phi\left(\mathbf{W}^{(2)}\mathbf{h}^{(1)}+\mathbf{b}^{(2)}\right)$, $\ldots$, $\mathbf{y} = f^{(L)}\left(\mathbf{h}^{(L-1)}\right) = f^{(L)}\circ\cdots\circ f^{(1)}(\mathbf{x})$
- **Choice of Last Layer Activation Func** Regression: $\mathbf{y} = f^{(L)}\left(\mathbf{h}^{(L-1)}\right) = \left(\mathbf{w}^{(L)}\right)^T\mathbf{h}^{(L-1)}+b^{(L)}$; Binary Classification: $\mathbf{y} = f^{(L)}\left(\mathbf{h}^{(L-1)}\right) = \sigma\left(\left(\mathbf{w}^{(L)}\right)^T\mathbf{h}^{(L-1)}+b^{(L)}\right)$
- **Back Propagation** Suppose $\mathcal{L}$ what I want to optimize, then for some variable $\mathbf{w}$ that we want to optimize w.r.t., $\frac{\partial\mathcal{L}}{\partial\mathbf{w}} =: \overline{\mathbf{w}}$
- **Back Prop Cost** Forward: one add-multiplicity operation per weight; Backward: two add-multiplicity operations per weight $\implies$ the Backward pass is about as expensive as two Forward passes. (cost is linear in # of layers, quadratic in # of units per layer)

Statistic on Samples

- **Sample Mean** $\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)}$. $\hat{\boldsymbol{\mu}}$ roughly quantifies where your data is located in space
- **Sample Cov** $\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}\right)\left(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}\right)^\top$ quantifies the shape of spread of the data

Euclidean projection Let $\mathcal{S}$ denote the subspace with dim $= k$ that is spanned by the basis $\{\mathbf{u}_1, ..., \mathbf{u}_K\} \subseteq \mathbb{R}^D$. Then,

- Any vector $\mathbf{y} \in \mathcal{S}$ could be represented as $\mathbf{y} = \sum_{i=1}^{K} z_i \mathbf{u}_i$, for some $z_1, ..., z_k \in \mathbb{R}$
- The projection of $\mathbf{x}$ onto $\mathcal{S}$ is given as $\text{Proj}_{\mathcal{S}}(\mathbf{x}) = \sum_{i=1}^{K}(\mathbf{x}^\top \mathbf{u}_i)\mathbf{u}_i = \sum_{i=1}^{K} z_i \mathbf{u}_i$ where $z_i = \mathbf{x}^\top \mathbf{u}_i$

Principle Component Analysis - Projection onto Subspace

- Let $\{\mathbf{u}_k\}_{k=1}^{K}$ be an orthonormal basis of the subspace $\mathcal{S}$.
- Define $\mathbf{U}$ to be a matrix with columns $\{\mathbf{u}_k\}_{k=1}^{K}$ then $\mathbf{z} = \mathbf{U}^T(\mathbf{x} - \hat{\boldsymbol{\mu}})$. Here the $\mathbf{z}$ is called the code vector
- Also, $\tilde{\mathbf{x}} = \hat{\boldsymbol{\mu}} + \mathbf{U}\mathbf{z} = \hat{\boldsymbol{\mu}} + \mathbf{U}\mathbf{U}^T(\mathbf{x} - \hat{\boldsymbol{\mu}})$ is called the reconstruction of $\mathbf{x}$
- Note: $\mathbf{U}\mathbf{U}^T$ is the projector matrix and $\mathbf{U}^T\mathbf{U} = I$ is the identity matrix.
- $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are both in $\mathbb{R}^D$ while $\tilde{\mathbf{x}}$ lives in a low dimensional subspace in $\mathbb{R}^D$. The code vector $\mathbf{x}$ is in $\mathbb{R}^K$, and is the low dim representation of the vector $\mathbf{x}$

PCA - Learning Subspace

- **Criteria I:** Minimize the reconstruction error: Find vectors in a subspace that are closest to data points, $\min_{\mathbf{U}} \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\right\|^2$
- **Criteria II:** Maximize the variance of reconstructions: Find subspaces where data has the most variability, $\max_{\mathbf{U}} \frac{1}{N}\sum_i\left\|\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}}\right\|^2$
- **Proof: Criteria I** $\equiv$ **Criteria II**; It suffices to show that $\frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\right\|^2 = \text{ const } - \frac{1}{N}\sum_i\left\|\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}}\right\|^2$

Support Vector Machines

- **Hinge Loss** is defined as $\mathcal{L}_{z,t} := \max\{0, 1 - zt\}$, where $z := \mathbf{w}^T\mathbf{x} + b$ and t is the target
- **Formulation** $\text{minimize}_{\mathbf{w},b} \sum_{i=1}^{N} \max\{0, 1 - t^{(i)}z^{(i)}(\mathbf{w}, b)\}$
- **L2 - Regularized** $\text{minimize}_{\mathbf{w},b} \sum_{i=1}^{N} \max\{0, 1 - t^{(i)}z^{(i)}(\mathbf{w}, b)\} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$
- **Optimal Separating *Hyperplane*** A hyperplane that separate two classes and maximizes the distance to the closest point from either class, i.e., maximizes the *margin* ($C = \frac{1}{\|\mathbf{w}\|_2}$) of the classifier.
- **Note:** A separating hyperplane is optimal *iff* it touches three data points near the margin.

AdaBoost Key Concepts

- **Boosting:** Train classifier sequentially, each time focusing on training data points that were previously misclassified.
- **Weighted Training Set:** The weighted misclassification rate is $\sum_{i=1}^{N} w^{(n)}\mathbb{I}(h(x^{(n)} \neq t^{(n)}))$ where $w^{(n)} > 0$ and $\sum_n w^{(n)} = 1$
- **Weak Learner (Informal):** Weak learners are algorithms that output slightly better than chance
- **Efficient Weak Learners:** We are interested in weak learners that are computationally efficient, for example decision trees, or even decision stumps (decision trees with only one split)
- **Weak Classifier:** The error of classifier h according to the given weights $\{w^{(1)}, ..., w^{(N)}\}$ is $err := \sum_{n=1}^{N} w^{(n)}\mathbb{I}(h(x^{(n)} \neq t^{(n)})) < \frac{1}{2} - \gamma$ for some $\gamma > 0$ ("better than chance")
- **Reduced Bias** AdaBoost reduces bias by making each classifier focus on previous mistakes.

AdaBoost Workflow

1. At each iteration we re-weight the training samples by assigning larger weights to samples (data points) that were classified incorrectly.
2. We train a new weak classifier based on the re-weighted samples
3. We add this weak classifier to the ensemble of weak classifiers. This ensemble is our new classifier.
4. Repeat

AdaBoost Algorithm

- Input data $\mathcal{D}_N = \{\mathbf{x}^{(n)}, t^{(n)}\}_{n=1}^{N}$ where $t^{(n)} \in \{-1, 1\}$
- A classifier (hypothesis $h : \mathbf{x} \to \{-1, 1\}$), and 0-1 loss $\mathbb{I}[h(x^{(n)}) \neq t^{(n)}] := \frac{1}{2}(1 - h(x^n) \cdot t^{(n)})$
- A classification procedure that returns a classifier h, e.g. best decision stump from a set of classifier $\mathcal{H}$, e.g. all possible decision stumps)
- Output a master classifier $H(x)$
- For $t = 1, ..., T$ (T is the number of iteration)

  1. Fit a classifier to data using weighted samples $h_t \leftarrow WeakLearn(\mathcal{D}_N, \mathbf{w})$. For example we can use
  $$h_t \leftarrow \arg\min_{h\in\mathcal{H}} \sum_{n=1}^{N} w^{(n)}\mathbb{I}\{h(\mathbf{x}^{(n)} \neq t^{(n)}\}$$
  2. Compute the weighted error
  $$\text{err}_t = \frac{\sum_{n=1}^{N} w^{(n)}\mathbb{I}\left\{h_t\left(\mathbf{x}^{(n)}\right) \neq t^{(n)}\right\}}{\sum_{n=1}^{N} w^{(n)}}$$
  3. Compute classifier coefficient
  $$\alpha_t = \frac{1}{2}\log\frac{1 - \text{err}_t}{\text{err}_t} \quad (\in (0, \infty))$$
  4. Update data weights
  $$w^{(n)} \leftarrow w^{(n)}\exp\left(-\alpha_t t^{(n)} h_t\left(\mathbf{x}^{(n)}\right)\right)\left[\equiv w^{(n)}\exp\left(2\alpha_t\mathbb{I}\left\{h_t\left(\mathbf{x}^{(n)}\right) \neq t^{(n)}\right\}\right)\right]$$

- Return $H(x) = \text{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(\mathbf{x})\right)$

AdaBoost Weighting Intuition

- $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$ where $\alpha_t = \frac{1}{2}\log\frac{1-\text{err}_t}{\text{err}_t}$ is "how much you trust weak learner t".
- Weak classifiers which get lower weighted error get more weight in the final classifier. When some err_t is small, $\alpha_t = \frac{1}{2}\log\frac{1-err_t}{err_t}$ is large in the final classifier.
- Also in weight updates, $w^{(n)} \leftarrow w^{(n)} \exp\left(2\alpha_t \mathbb{I}\left\{h_t\left(\mathbf{x}^{(n)}\right) \neq t^{(n)}\right\}\right)$. **If** $err_t \approx 0$ **then**, α_t hight so misclassified examples get more attention. **Else If** $err_t \approx 0.5$ **then** α_t low hence misclassified examples not emphasized.

AdaBoost Geometric Convergence and Generalization Errors

- **Theorem:** Assume that at each iteration of AdaBoost the WeakLearn returns a hypothesis with error $err_t < \frac{1}{2} - \gamma$ for all $t = 1, ..., T$ with $\gamma > 0$. The trainning error of the output hypothesis $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$ is at most
$$L_N(H) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left\{H\left(\mathbf{x}^{(i)}\right) \neq t^{(i)}\right\} \leq \exp\left(-2\gamma^2 T\right)$$
under the simplifying assumption that each weak learner is $\gamma > 0$ better than a random predictor. The convergence is called geometric convergence, very fast!
- **Generalization Error:** *AdaBoost's Training error converges to zero.* In testing set, AdaBoost can overfit when we add too much weak learners. However this doesn't always happen. **There are cases where the testing error keeps decreasing even when training error is zero.** WHY? This is because even when training error is zero, the margin (= sample distance to decision boundary) is still improved by further boosting iterations.

Additive Models - AdaBoost Alternative View Point

- Consider hypothesis class $\mathcal{H}$ with $\mathcal{H} \ni h_i : \mathbf{x} \to \{-1, 1\}$ weak learners. Aka bases in this context.
- An additive model with m terms is given by $H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x})$ where $(\alpha_1, ..., \alpha_m) \in \mathbb{R}^m$ (generally $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$)
- **Stage-wise Training of Additive Models**

```
Initialize $H_0(x) = 0$
For $m = 1$ up to $T$ do
    Compute $m$-th hypothesis $h_m = H_{m-1} + \alpha_m h_m$, i.e. $h_m$ and $\alpha_m$ assuming previous additive model $H_{m-1}$ is fixed
        $(h_m, \alpha_m) \leftarrow \underset{h \in \mathcal{H}, \alpha}{\text{argmin}} \sum_{i=1}^{N} \mathcal{L}\left(H_{m-1}\left(\mathbf{x}^{(i)}\right) + \alpha h\left(\mathbf{x}^{(i)}\right), t^{(i)}\right)$    (1)
    Add it to the additive model
        $H_m = H_{m-1} + \alpha_m h_m$
```

- **Additive Model with Exp Loss:** Consider the Exponential Loss $\mathcal{L}_E(z, t) := \exp(-tz)$. Then, (1) becomes
$$(h_m, \alpha_m) \leftarrow \underset{h \in \mathcal{H}, \alpha}{\text{argmin}} \sum_{i=1}^{N} \exp\left(-\left[H_{m-1}\left(\mathbf{x}^{(i)}\right) + \alpha h\left(\mathbf{x}^{(i)}\right)\right] t^{(i)}\right) = \sum_{i=1}^{N} \exp\left(-H_{m-1}\left(\mathbf{x}^{(i)}\right) t^{(i)} - \alpha h\left(\mathbf{x}^{(i)}\right) t^{(i)}\right)$$
$$= \sum_{i=1}^{N} \underbrace{\exp\left(-H_{m-1}\left(\mathbf{x}^{(i)}\right) t^{(i)}\right)}_{\triangleq w_i^{(m)}} \exp\left(-\alpha h\left(\mathbf{x}^{(i)}\right) t^{(i)}\right) = \sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h\left(\mathbf{x}^{(i)}\right) t^{(i)}\right)$$
- **h Optimized at:** $h_m \leftarrow \underset{h \in \mathcal{H}}{\text{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h\left(\mathbf{x}^{(i)}\right) t^{(i)}\right) \equiv \underset{h \in \mathcal{H}}{\text{argmin}} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\left\{h\left(\mathbf{x}^{(i)}\right) \neq t^{(i)}\right\}$
- **Weight Update Optimized at:** $w_i^{(m+1)} \leftarrow w_i^{(m)} \exp\left(-\alpha_m h_m\left(\mathbf{x}^{(i)}\right) t^{(i)}\right)$
- **Coefficient Optimized at:** $\alpha = \frac{1}{2}\log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ where $err_m \triangleq \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\left\{h_m\left(\mathbf{x}^{(i)}\right) \neq t^{(i)}\right\}}{\sum_{i=1}^{N} w_i^{(m)}}$

Naïve Bayes

- **Naïve Assumption:** Naïve Bayes assumes that the features are *conditionally independent given the class* c, i.e. $p(c, x_1, ..., x_D) = p(c)p(x_1|c)...p(x_D|c)$. **Benefit:** This gives us a compact representation of the joint distribution. $\mathcal{O}(2^{D+1} - 1) \to \mathcal{O}(2D + 1)$
- **Bayes Rule** $p(c|\mathbf{x}) = \frac{p(\mathbf{x}, c)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|c)p(c)}{p(\mathbf{x})}$ **Formally** posterior $= \frac{\text{Class likelihood} \times \text{prior}}{\text{Evidence}}$
- **Naïve Bayes Inference** $p(c|\mathbf{x}) = \frac{p(c)p(\mathbf{x}|c)}{\sum_{c'} p(c')p(\mathbf{x}|c')} = \frac{p(c)\prod_{j=1}^{D} p(x_j|c)}{\sum_{c'} p(c')\prod_{j=1}^{D} p(x_j|c')}$ **Shorthand** $p(c|\mathbf{x}) \propto p(c)\prod_{j=1}^{D} p(x_j|c)$
- **Computational Cost of Naïve Bayes:** (1) **Training Time:** estimate parameters using MLE, requires one pass in the data. (2) **Test Time:** apply Baye's Rule. Cheap because of the model structure.

Bayesian Parameter Estimation

- **Posterior Distribution:** $p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta})}{\int p(\boldsymbol{\theta}')p(\mathcal{D}|\boldsymbol{\theta}')\mathrm{d}\boldsymbol{\theta}'}$
- **Gamma As Prior:** $p(\theta; a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1}$. **Proportionality:** $p(\theta; a, b) \propto \theta^{a-1}(1-\theta)^{b-1}$
- **Maximum A-posteriori Estimation:** $\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}) = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \log p(\mathcal{D}|\boldsymbol{\theta})$

Properties of Gaussian Distribution

- $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is defined as $p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$
- **Empirical Mean** $\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)}$ **Empirical Cov** $\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}\right)\left(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}\right)^\top$

Gaussian Discriminant Analysis (Gaussian Bayes Classifier)

- **Idea:** Make decisions by comparing class posteriors. $\log p\left(t_k|\mathbf{x}\right) = \log p\left(\mathbf{x}|t_k\right) + \log p\left(t_k\right) - \log p(\mathbf{x})$
- **Expanded** $\log p\left(t_k|\mathbf{x}\right) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log\left|\boldsymbol{\Sigma}_k^{-1}\right| - \frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_k\right)^T \boldsymbol{\Sigma}_k^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_k\right) + \log p\left(t_k\right) - \log p(\mathbf{x})$
- **Decision Boundary:** $\log p\left(t_k|\mathbf{x}\right) = \log p\left(t_l|\mathbf{x}\right) \implies \left(\mathbf{x}-\boldsymbol{\mu}_k\right)^T \boldsymbol{\Sigma}_k^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_k\right) = \left(\mathbf{x}-\boldsymbol{\mu}_\ell\right)^T \boldsymbol{\Sigma}_\ell^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_\ell\right) + C_{k,l}$
  Then, $\mathbf{x}^T\boldsymbol{\Sigma}_k^{-1}\mathbf{x} - 2\boldsymbol{\mu}_k^T\boldsymbol{\Sigma}_k^{-1}\mathbf{x} = \mathbf{x}^T\boldsymbol{\Sigma}_\ell^{-1}\mathbf{x} - 2\boldsymbol{\mu}_\ell^T\boldsymbol{\Sigma}_\ell^{-1}\mathbf{x} + C_{k,l}$
- **Decision Boundary:** is quadratic since gaussian is quadratic. When we have to humps that share the same covariance, the decision boundary is linear.
- **VS Logistic Regression** (1) GDA is generative while LR is discriminative model. (2) GDA makes stringer modelling assumptions: assumes gaussian distributon. When assumption true, GDA asymptotically efficient. (3) LR more robust, less sensitive to incorrect modelling assumptions (LR uses CE, no assumption.) (4) Class-conditional distributions usually lead to logistic classifier.